Experiences in Teaching a Specialty Multicore Computing Course

Peter Strazdins
Computer Systems Group,
Research School of Computer Science,
The Australian National University

Intel Corporation
Santa Clara, 08 May 2013

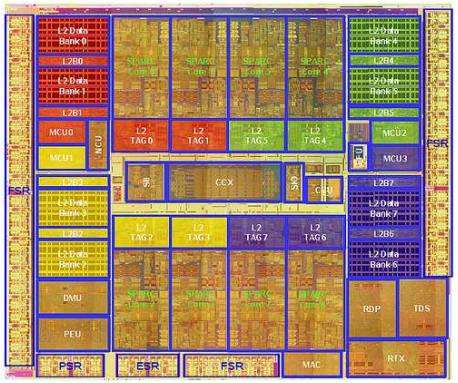
(slides available from http://cs.anu.edu.au/~Peter.Strazdins/seminars)

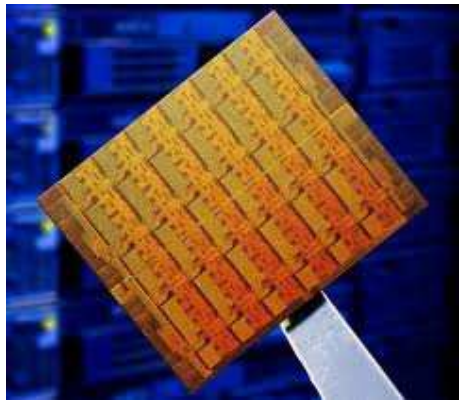
THE AUSTRALIAN NATIONAL UNIVERSITY

1 Overview

comp8320 Multicore Computing: Principles and Practice:

- course context in computer systems curriculum
- design philosophy
- course structure: modules, assessment scheme
- teaching and learning experiences
  - programming paradigms
  - architectural insights
  - infrastructure support: problem domain & platform related
- conclusions and future work





2 Context in the ANU Computer Systems Curriculum

course for postgraduates and advanced undergraduates:

- assumed knowledge:
  - comp2300 Introduction to Computer Systems: illustrative and contemporary processors
  - comp2310 Concurrent and Distributed Systems
- related pre-existing courses:
  - comp3320 High Performance Scientific Computation: data modelling, programming & performance issues
  - comp4300 Parallel Systems: practical issues in shared & distributed memory (now includes TBB)

  Only offered every 2nd year $\Rightarrow$ can't use as prerequisites!
- Multicore Computing must be a specialty course
  - a 'capstone', with the most advanced aspects of architecture, concurrency and performance evaluation

3 Design Philosophy

- aim: prepare advanced students for the rapidly unfolding future of multi-core / manycore
- educational approaches:
  - research-based education: relevant research and practice of our group
  - cognitive apprenticeship: pass on instructor's experiences when students undertake similar activities
- key goal: teach how architectural effects relate to changes in program performance
  - sophisticated infrastructure must be provided for students to explore this
  - strong emphasis on the use of software tools (e.g. profilers)

  Note: the Intel Single-Chip Cloud Computer (SCC) was added for 2011
  - no cache coherency $\Rightarrow$ cores are separate nodes, communication via messages!

4 Module-based Course Structure

	module (2 hour lecture)	tut.	lab.	ass.
1	Advent of Multicore	1	1	
2	Multicore Architecture and the T2	2	2	1 (20%)
3	Advanced OpenMP Programming		3	1
4	Performance Issues; Synchronization	3	4	1
5	Software Engineering for Multicore	4	5	
6	Operating System Issues and Virtualization		6	
7	Graphics Processing Units	5	7	2 (15%)
8	On-chip Networks & the Single-chip Cloud Computer	6	8	3 (10%)
9	Trans. Mem., Speculation, Heterogeneous Cores	7		
10	Outlook (Manycore) and Review	8		

- main references: 3 textbooks, 3 ‘slide-sets’ & various papers
- modules 1–7 scheduled in 1st half of semester
- 2009: small group mini-projects replaced modules 6, 8, 10; ass. 2 & 3
  - problematic: high workload, difficult to examine material
- 2011: worked well; module 5 least popular, needed more time on 7 & 8

5 Experiences: Programming Paradigms

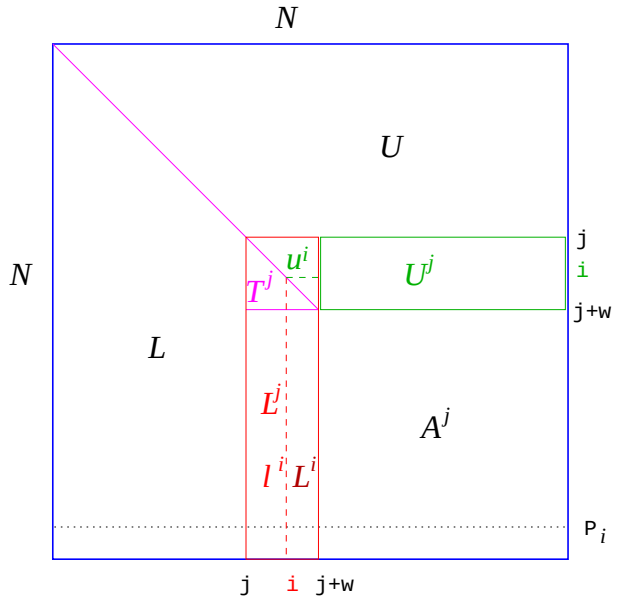
- shared memory (OpenMP), device (CUDA), and message passing (RCCE)!
- students picked up CUDA easily from prior experience in OpenMP, e.g.

```
reverse <<<1, N/2>>> (a_d, N);
...
__global__ void reverse(int *a, int N)
{ int idx = threadIdx.x;
  int v = a[N-idx-1];
  a[N-idx-1] = a[idx]; a[idx] = v;
}
```

```
#pragma omp parallel num_threads(N/2) \
  default(shared)
{ int idx = omp_get_threads_num();
  int v = a[N-idx-1];
  a[N-idx-1] = a[idx]; a[idx] = v;
}
```

- 3 assignments were based on single theme (LINPACK)
  - √ commonality in experience (and infrastructure)
  - × lack of prior familiarity, tricky, started to “get sick” of one application
- more time was needed for RCCE; programming exercises in ass. 3 had to be limited
- programmability (design, impl. & debugging) of each paradigm
  - student consensus: CUDA was the hardest!

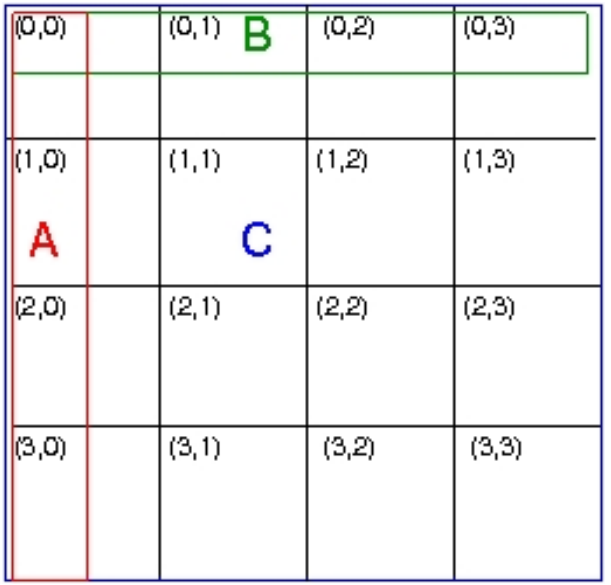
6 Experiences: Architectural Insights (from LINPACK)



- LINPACK on the UltraSPARC T2: fixed decomposition scheme
  - what was the main cause for performance loss for ≥ 32 threads?
  - by using profiling tools (Solaris analyzer), most students inferred was due to destructive sharing
    - inferable for an increasing amount of time in barriers
- LINPACK on CUDA: main task was a robust matrix multiply kernel
  - instructor-provided kernel caused error on subsequent `cudaFree();`
  - over-write suspected, but not detected when kernel was rigorously tested in isolation (???)
  - 1 student solved this 'challenge problem': over-read!

7 Experiences: Architectural Insights (SCC)

- LINPACK: only components (e.g. matrix multiply implemented
- *vary $P \times Q$ grid shape & explain the performance*
  - SCC's 48 cores gave plenty of ratios!
  - most explained correctly the counter-intuitive result that near-square are best (why?)



- *explain effect of r repetitions* (1×8 grid, $1000 \times 1000 \times 48$ multiply)

broadcast	RCCE	ring	tree
MFLOPS ($r = 1$)	107	106	140
MFLOPS ($r = 10$)	164	170	172

  - was too subtle for students – suggested cache effects
  - experiments on a 1×1 grid negated this!
- lack of any profiling tools for SCC cores made this hard

8 Infrastructural Support - Problem Domain Related

- principal learning activity was optimizing components and varying parameters, analyzing their effects
- required sophisticated test programs, e.g.

```
runXe ./linpack [-p] [-b NB] [-w W] [-v v] [-k] [-a info] N
```

  with support for debugging and rigorous correctness checking
- for CUDA, isolated test matrix multiply test program also needed
  - debugging by printing ($-\mathrm{p}$) only useful if $a_{i,j} = r_i i + r_j j$
  - but for GPU ($N \leq 8192$), roundoff errors dwarfed subtle alg. errors!
  - solution: limit the result to exact integers, by setting matrices as:

$$a_{i,j} = r_i(i\%M(r_i, \mathrm{NB})) + r_j(j\%M(r_j, \mathrm{NB}))$$

$$M(r,k) = \frac{2^{w/2}}{(r+1)k}$$ where w is the mantissa width

- for SCC, checking result against serial algorithm took > 1 minute!
  - solution: fast (parallel) prediction for the above scheme

9 Infrastructure Support: Platform Related

- need: to take a large number of measurements reliably (mutual exclusion) and efficiently (few seconds)
  - students are used to having dedicated resources whenever they want them!
- for GPUs, used NCI's Xe cluster: access via batch system
  - soon becomes unwieldy! solution `runXe` pseudo-interactive script
- the SCC is a single user machine: all process on cores run as `root`!
  - only 1 processor per core can safely access message passing buffers
  - interfering jobs lock up machine (eventually bring it down!)
- solution: provide a submission script ensuring safe, exclusive access
  - overall and per-core lockfiles with timeouts were not robust enough
  - needed to be combined with seek-and-purge of non-system processes on cores

10 Conclusions

- key learning goal of understanding how architectural effects relate to changes in program performance
- modular structure with programming and performance analysis activities well supported this
- suitable infrastructure (highly sophisticated, instrumented test programs and job control scripts) was needed
  - possible for course-work students to even use SCC safety & efficiently (received MARC SCC Recognition for this)
- students with a general computer systems background, could meet the courses' learning objectives (including learning 3 different programming paradigms)
  - caveat: more time is needed for learning message passing
  - correctly interpreting performance data remains difficult without tools
- full course details including infrastructure are freely available at http://cs.anu.edu.au/student/comp8320

11 Future Work

- the Multicore Computing course will next run in 2014
  - course title may need an update!
- will need to develop material for new platforms
  - the UltraSPARC T2 suffered a disk crash, cannot fully restore :(
    (a little dated anyway, but highly threaded CMP still relevant)
  - the Intel SCC may not be available . . .
    (unclear if programming model will reflect future chips)
- still aim to provide state-of-the-art experiences to the students!
- reconsider role of multi/manycore in the High Performance Scientific Computation and Parallel Computing courses as well

Acknowledgments!

- Sun Microsystems (now Oracle) for the donation of the Ultra-SPARC T2
- Intel Corporation for the donation of the Single-chip Cloud Computer
- NCI National Facility for support and usage of the Xe GPU cluster

Questions???